Universidad Zaragoza

Trabajo Fin de Grado

Integración de un sistema de aprendizaje automático en un motor de renderizado
Integration of a machine learning system in a rendering engine

Autor

Miguel Ángel Cosculluela Gracia

Director

Julio Marco Murria

Ponente

Diego Gutiérrez Pérez

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2019

AGRADECIMIENTOS

Quiero dar las gracias especialmente a Julio por dirigirme este proyecto y aguantar todas mis preguntas y problemas y tener siempre una respuesta. A Ibón por su ayuda con Mitsuba que tantos problemas me dio y dar las gracias a todo el *Graphics and Imaging Lab* por hacerme sentir como en casa e incluirme en sus rutinas como a uno más.

Agradecer también a mis amigos Fran, Rubén y Julián por estar siempre ahí y apoyarme a lo largo de todos estos años.

Y finalmente agradecer a mis padres y hermano por animarme durante la carrera y confiar en mi.

RESUMEN

El renderizado o generación de imágenes sintéticas basado en físicas es un campo de la informática gráfica muy complejo que requiere simular el transporte de luz y sus interacciones con medios y geometrías. Este proceso es muy costoso de realizar para la obtención de imágenes de alta calidad por lo que existe una fuerte investigación en nuevos o mejores métodos de generarlas. El El uso de técnicas de aprendizaje autmático basadas en redes neuronales ha demostrado gran potencial en investigación aplicada al renderizado, sin embargo su uso no es trivial, las distintas librerías que las desarrollan son ajenas a los sistemas de render y no están preparadas para hacer un uso conjunto.

En este proyecto se presenta un nuevo sistema diseñado y desarrollado con el fin de facilitar el uso de librerías de redes neuronales dentro de un motor de renderizado. Para ello se realiza un estudio de las características de los renderizadores más utilizados actualmente en investigación actuales junto con otro sobre las librerías de aprendizaje automático de uso más extendido. Escogiendo finalmente el renderizador Mitsuba y la librería de aprendizaje automático *LibTorch* se realiza un análisis de sus flujos de ejecución. Culminando en el diseño de una arquitectura que integra LibTorch en Mitsuba.

El sistema propuesto es validado mediante la reproducción en el sistema desarrollado de un artículo de reducción de varianza en renderizado desarrollado por Disney para la aplicación en sus peliculas de animación. Finalmente sobre el artículo se propone e implementa una variación aplicando el método a imágenes con medios participativos.

Índice

Capítulo 1

Introducción

1.1. Objetivo y contexto del trabajo

La informática gráfica es el campo que estudia la generación de imágenes y su modificación mediante técnicas computacionales. Estas técnicas son por ejemplo el renderizado de imágenes que las genera a partir de escenarios virtuales o la modificación de éstas mediante herramientas de edición.

Concretamente el renderizado refiere a un conjunto de técnicas que generan imágenes sintéticas a partir de modelos geométricos, luces, cámaras etc. Estas técnicas tienen mucho recorrido y llevan siendo estudiadas desde los años 80. El renderizado basado en físicas realiza este proceso basándose en modelos físicamente correctos del transporte de luz. Este proceso es muy costoso de generar debido a que calcular de forma precisa la luz en una escena requiere calcular todas las interacciones que produce con la geometría, sus materiales o los medios que interaccionan con ella. Esto deriva en tiempos de procesamiento muy largos. Por ejemplo, en la película de Disney, Frozen, en la figura 1.1 se puede ver un ejemplo de una escena de la película, una imagen compleja costó 30 horas de procesamiento en un cluster de 4000 ordenadores. Por ello en investigación se estudian nuevos métodos para conseguir imágenes mas realistas y de mas calidad en un tiempo menor.

Clásicamente para avanzar en este campo se utilizan técnicas analíticas y estocásticas apoyadas en algunos casos por métodos de aprendizaje automático. Sin embargo, con los recientes avances en aprendizaje profundo, cada vez se están presentando más proyectos de investigación en los que se están usando técnicas basadas en redes neuronales. Esta técnica ha demostrado una gran utilidad y flexibilidad permitiendo resolver multitud de problemas que existen en renderizado. Por ejemplo se ha modelado la dispersión de la luz en nubes *Kallweit et al.*[2] o la compresión de

Figura 1.1: Castillo de hielo de la pelicula Frozen [1]

texturas *Rainer et al.*[3]. En la figura 1.2 se puede observar un resultado original del artículo de *Kallweit et al.*[2] que presenta dicho método. Como se puede observar el uso de redes neuronales consigue resultados equivalentes a los de referencia.

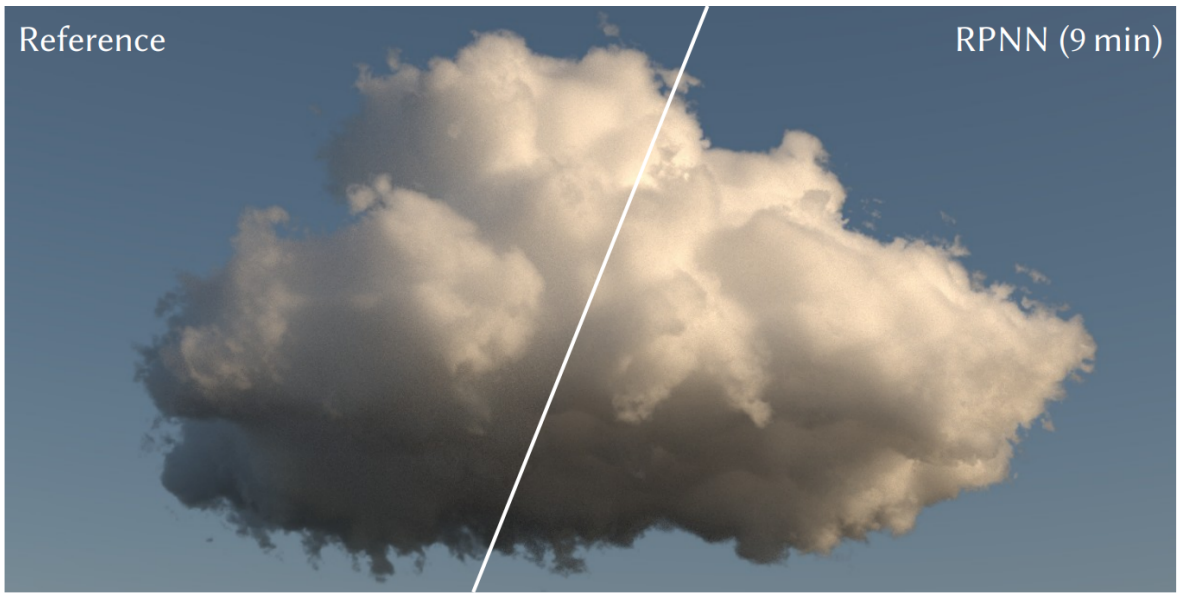

Figura 1.2: Izquierda: Imagen de nube de referencia. Derecha: Nube generada con el método de *Kallweit et al.*[2]. La imagen ha sido obtenida de dicho artículo.

Utilizar técnicas de aprendizaje automático dentro de render es problemático porque las librerías de ambos campos se desarrollan de forma separada. Estos no están preparados para su uso conjunto debido a que se han desarrollado de forma independiente por ejemplo. tienen flujos de ejecución muy diferentes y no poseen los mismos tipos de datos. Por ello en este trabajo se realiza una integración de ambas partes, estudiando que opciones hay disponibles de cada una y proponiendo un diseño de un sistema que permita hacer uso de render con redes neuronales de forma directa

mejorando su usabilidad. Las tareas realizadas en este trabajo son:

- Estudio de los distintos motores de render existentes seleccionando aquel que mas versatilidad ofrecía.
- Estudio de las distintas librerías de aprendizaje automático disponibles que compartían lenguaje con el motor de render escogido previamente y cuya interfaz es mas avanzada.
- Estudio de las características e interacciones posibles entre renderizador y librería de aprendizaje automático.
- Diseño de sistema que atienda a dichas interacciones y características integrando ambas partes en una.
- Prueba del sistema desarrollado con un problema real mediante la replicación de un artículo de investigación de Disney Research sobre eliminación de varianza en imágenes renderizadas.
- Estudio y realización de una variación sobre el método del artículo aplicándolo al renderizado de medios participativos.

1.2. Organización

Primero se explicará en el capítulo 2 el marco teórico dando los conocimientos básicos para poder entender este proyecto, dando una breve motivación y explicando la problemática que contiene. En el capítulo 3 se presenta un análisis de las distintas opciones de renderizadores y librerías de aprendizaje automático que existen para desarrollar el proyecto así como el diseño implementado. En el capítulo 4 se presenta un artículo de investigación que se reproduce usando el sistema antes diseñado. Finalmente en el capítulo 5 se describen las conclusiones y posible trabajo futuro.

1.3. Planificación

El proyecto ha tenido una duración total de 6 meses con un tiempo estimado de 350 horas, desde enero de 2019 a junio del mismo año. La búsqueda de bibliografía es un proceso que se realizó durante todo el trabajo incluyendo la búsqueda de información sobre renderizadores y librerías de aprendizaje automático así como la lectura de artículos del estado del arte de render que hacen uso de redes neuronales. La implementación del sistema fue la parte que más tiempo consumió debido a la necesidad de asegurar un correcto funcionamiento en todos los escenarios de ejecución.

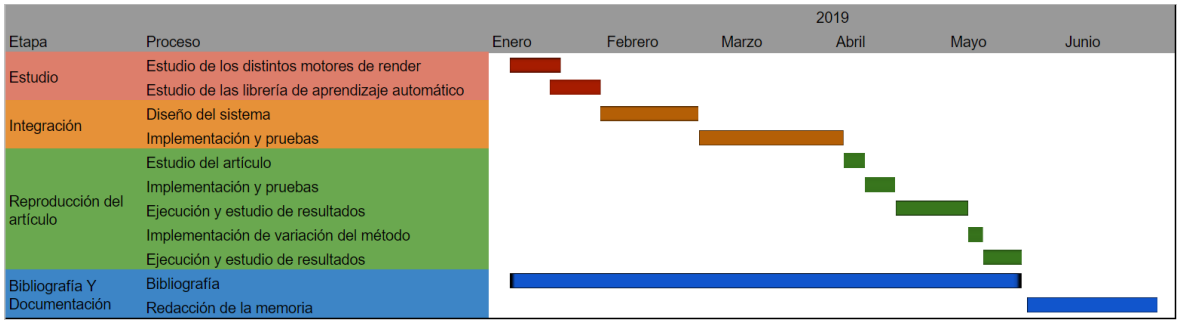

Figura 1.3: Diagrama de Gantt. Se refleja la distribución del trabajo a lo largo del tiempo.

Capítulo 2

Marco teórico

En este capítulo se introducen todos los conceptos teóricos necesarios para entender este trabajo. Se abordarán dos temas principales, la teoría detrás de la técnica de renderizado y una teoría básica sobre redes neuronales. Posteriormente se tratará la problemática que hay en el uso de estas dos tecnologías juntas.

2.1. Renderizado de imágenes

El renderizado de imágenes basados en físicas es un proceso que busca simular los eventos físicos que ocurren en el transporte de luz en un escenario real y la captura de éste por una cámara virtual generando una imagen. Este proceso es muy costoso de calcular con precisión debido a la interacción de luz con la geometría de las escena, la características de los materiales o si hay medios participativos como líquidos o niebla.. Todas estas partes añaden complejidad a este campo y suponen nuevas posibilidades de investigación en el desarrollo de algoritmos de renderizado.

En la actualidad este campo de la informática gráfica es aplicado y estudiado tanto en industria como en academia. En la industria audiovisual, concretamente en el cine y en los videojuegos estas técnicas están bien asentadas, su uso también se extiende a industrias como la imagen médica, la arquitectura, o el prototipado de producto. En el cine no solo se usa para la realización de películas de animación sino también como efectos especiales en las películas convencionales. Un claro ejemplo de este uso es la creación de escenas con personas fallecidas o rejuvenecidas como se ha usado en las recientes películas de *Star Wars* o las películas hiperrealistas hechas casi en su totalidad de imágenes renderizadas como Gravity o El libro de la selva (2016). Por otra parte en la industria de la decoración también es un elemento muy útil, Ikea realiza el 75 % de sus imágenes de catalogo mediante el renderizado. En el ámbito empresarial hay múltiples empresas que desarrollan sus motores de renderizado basados en físicas como Disney, Weta Digital, Autodesk y Corona.

2.1.1. La ecuación de render

El proceso de simulación del transporte de luz en medios no participativos se realiza asumiendo que el transporte se realiza sobre los límites de la geometría. La luz se calcula integrando la proveniente de todas las direcciones y que llegan a un punto de la geometría. Este proceso fue modelado por *Kajiya*[4] mediante la ecuación de render 2.1.

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega L_i(x, \omega_i) f_r(x, \omega_i, \omega_o)(n \cdot \omega_i) d\omega_i \tag{2.1}$$

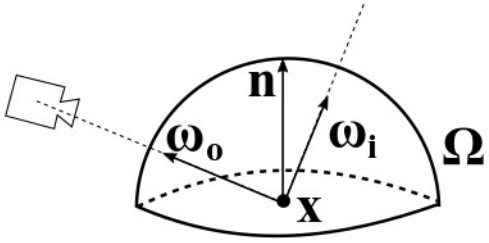

Figura 2.1: Hemiesfera, imagen provisional

Esta ecuación modela la radiancia saliente $L_o(x, \omega_o)$ en un punto x en dirección ω_o. $L_e(x, \omega_o)$ indica la radiancia emitida por la superficie. La integral calcula la radiancia que recibe el punto ($L_i(x, \omega_i)$) en la hemiesfera Ω, en la figura 2.1 puede verse un esquema del cálculo de la radiancia en dicha hemiesfera. El término f_r modela una función de distribución de reflectancia bidireccional o BRDF por sus siglas en inglés e indica cuanta de la radiancia que recibe el material es reflejada en la dirección ω_o. Pueden verse ejemplos de BRDFs en la figura 2.2. El término final tiene en cuenta la geometría entre el rayo de luz incidente y la normal de la superficie en x. Esta ecuación no puede ser resuelta analíticamente debido a la geometría arbitraria que hay alrededor de cada punto. Para su resolución se requiere de técnicas de aproximación.

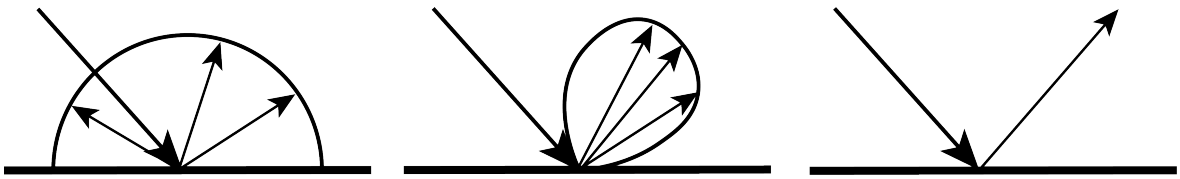

Figura 2.2: Diferentes modelos de BRDFs: reflexión Lambertiana (izquierda), reflexión muy direccional y especular (centro), y reflexión especular perfecta.

2.1.2. Medios participativos y transferencia radiativa

Los medios participativos incluyen una nueva dificultad a la hora de calcular el transporte de luz en una escena, debido a que anteriormente en la ecuación 2.1 solo se tienen en cuenta las interacciones entre superficies. Un medio participativo esta definido como un espacio en cual se producen distintos fenómenos, los cuales interaccionan con la luz sin necesidad de que exista una superficie. Estos fenómenos pueden ser de emisión, absorción y dispersión tanto entrante como saliente. En este trabajo solo nos centraremos en la absorción y la dispersión en medios. Estos dos fenómenos son caracterizados por medio de sus coeficiente de absorción σ_a y dispersión σ_d que modelan la probabilidad de realizar uno de estos fenómenos. Para calcular la transferencia radiativa del medio se hace uso de la ecuación de transferencia radiativa *chandrasekhar*[5]:

$$L(x, \omega_o) = T_r(x \leftrightarrow x_p)\ L_o(x_p, \omega_o) + \int_{s_0}^{s1} T_r(x \leftrightarrow x_s) L_i(x_s, \omega_o)\ ds \tag{2.2}$$

donde $T_r(x \leftrightarrow x_p)$ modela la transmitancia en el medio (cantidad de luz que no se ha extinguido dentro del medio entre x y x_p) y $L_o(x_p, \omega_o)$ modela la luz incipiente en el medio (ecuación 2.1). Estos dos términos modelan la absorción. Por otro lado, la integral definida entre s_0 y s_1 modela la dispersión de la luz teniendo en cuenta también la transmitancia junto con la luz dispersada en un punto hacia una dirección $L_i(x_s, \omega_o)$.

La luz que ha dispersado el medio se puede modelar de la siguiente forma:

$$L_i(x, \omega_o) = \sigma_s(x) \int_{\Omega_{4\pi}} \rho(x, \omega_i, \omega_o)\ L(x, \omega_i)\ d\omega_i \tag{2.3}$$

siendo $\rho(x, \omega_i, \omega_o)$ la función de fase la cual modela la dirección hacia la cual se produce la dispersión en el medio y la integral sobre $\Omega_{4\pi}$ modela el total de luz dispersada en cada dirección de la esfera posicionada en el punto x.

Finalmente, existen dos tipos de medios participativos: homogéneos y heterogéneos. Se diferencian en función de si su transmitancia es constante o variable en toda su extensión. Si $T_r(x \leftrightarrow x_p)$ es constante, se puede resolver de manera analítica como:

$$T_r(x \leftrightarrow x_p) = e^{-\sigma_t\ \|x - x_p\|} \tag{2.4}$$

donde σ_t es denominado coeficiente de extinción y se calcula como $\sigma_t = \sigma_a + \sigma_s$.

2.1.3. Monte Carlo

Como se ha comentado en las secciones 2.1.1 y 2.1.2 la ecuación de render y su equivalente para medios participativos no pueden ser resueltos de forma analítica debido

a que tienen en cuenta la geometría arbitraria de la escena y los medios pueden tener propiedades complejas. Para calcular estas ecuaciones se debe hacer uso de técnicas de aproximación como la técnica de integración estocástica por Monte Carlo, usando la aproximación

$$\int_{\Omega} f(x)\, dx \approx \frac{1}{N} \sum_{x_i \in \Omega}^{N} \frac{f(x_i)}{pdf(x_i)} \tag{2.5}$$

donde $pdf(x_i)$ es la distribución de probabilidad de $x_i \in \Omega$. El efecto que genera esta aproximación es la introducción de varianza o vías en la imagen. La primera se puede ver como ruido en la imagen y la segunda como una diferencia entre el valor real y el obtenido. Esto limita la calidad de las imágenes obtenidas.

Este método de aproximación es aprovechado en diferentes métodos de renderizado como *path tracing Veach*[6] que en vez de calcular toda la luz que llega a un punto desde todas las direcciones, solo calcula sobre una dirección generando de forma iterativa rebotes en distintos puntos de la geometría de la escena. De esta forma el número de muestras se traslada de cada punto a la repetición de caminos. Otros métodos como *bidirectional path tracing Lafortune et al.*[7] que es una variación de *path tracing* y que traza caminos desde el sensor de la cámara virtual y desde las fuentes de luz, de esta forma se pueden unir ambos caminos y obtener mejor resultado que con el algoritmo original.

2.2. Aprendizaje automático y redes neuronales

El aprendizaje automático es la rama de la inteligencia artificial enfocada en el desarrollo de técnicas que permitan aprender a un sistema a partir de la información que se le proporcione. En este campo se pueden distinguir dos grandes grupos de técnicas, aquellas que hacen uso de un aprendizaje supervisado y las que usan aprendizaje no supervisado. Dado que en este trabajo solo se hace uso de las primeras, a continuación se dará una explicación de ella.

En el aprendizaje supervisado se tiene un sistema capaz de aprender mediante comparación con un valor ideal. Los datos de entrenamiento están formados por parejas $\{X, Y\}$ siendo X el valor de entrada al sistema e Y la salida que queremos que aprenda. De esta forma un sistema que sea iterativo podrá ir adaptándose para acercarse cada vez más a esa salida ideal. Para que un entrenamiento sea considerado como bueno no solo debe tener un error bajo, sino que el sistema debe ser capaz de funcionar de una manera relativamente buena con datos que no ha visto nunca.

Las redes neuronales son un modelo computacional que permite aproximar funciones no lineales mediante técnicas de aprendizaje. Si bien no son nuevas con el avance surgido en los últimos años en hardware como GPUs, siendo éstas más potentes y con más memoria se ha vuelto mucho más práctico entrenar grandes redes neuronales de forma eficiente. Estos modelos han demostrado ser una gran herramienta en múltiples campos como la informática gráfica. Por ejemplo en el renderizado de imágenes permite abordar y resolver problemas que anteriormente debían ser tratados con técnicas analíticas. Algunos ejemplos recientes de su uso son: la compresión de texturas de *Rainer et al.*[3], simulación de la dispersión de la luz en nubes de *Kallweit et al.*[2], simluación de la dispersión bajo la superficie de materiales de *Vicini et al.*[8], muestreo de muestras para mejorar el renderizado por Monte Carlo de *Müller et al.*[9]. A nivel de industria también se esta usando como por ejemplo Disney o Nvidia con los trabajos de *Bako et al.*[10] y *Chaitanya et al.*[11] que están realizando investigación sobre la eliminación de varianza en imágenes, un problema muy común en render y que con técnicas rápidas permite ahorrar gran cantidad de tiempo.

2.3. Problemática

Como se ha comentado en la sección 2.3 hay multitud de posibilidades en el uso de redes u otros métodos de aprendizaje en render. Sin embargo su uso en este campo no es sencillo. Los sistemas de render y de aprendizaje se han desarrollado de forma separada. Por ello cuando se quieren hacer uso de forma conjunta supone un gran obstáculo que deriva en integraciones específicas para el problema que se esta resolviendo, y que poseen poca o ninguna posibilidad de reutilizarse.

Para entender mejor por qué es tan problemático usarlos de forma conjunta se debe observar como están estructurados estos sistemas.

Los motores de render renderizado utilizados habitualmente en investigación suelen estar formados por los siguientes componentes:

- Modelos de cámaras: Representan conjuntos de lentes y sensores de randiancia donde se genera la imagen.
- Modelos geométricos: Representan objetos del mundo real basados en modelos de primitivas como triángulos, esferas, planos, etc.
- Modelos de emisión de luz: Como luces de área o puntuales.
- Texturas y funciones de reflectancia (BRDFs). Sección 2.1.1.
- Filtros de imagen: Como filtros gaussianos o filtros de eliminación de varianza que permiten reconstruir una imagen en base a las muestras de radiancia que

llegan al sensor.

- Modelos de medios participativos
- Integradores: Implementan distintos métodos para resolver la ecuación de render y la ecuación de radiancia transitiva. Secciones 2.1.1, 2.1.2

Los sistemas de aprendizaje más comunes utilizados en investigación están formados por:

- Tensores: tipo de dato matricial capaz de representar datos de cualquier dimensión que almacena el grafo de operaciones que se le aplican.
- Algoritmos de descenso del gradiente: SGD *robbins et al.*[12], Adam *Kingma et al.*[13], AdaGrad *Duchi et al.*[14], etc.
- Distintos tipos de capas y funciones de activación.
- Distintas funciones de pérdida que permiten realizar la optimización mediante el cálculo del error entre la referencia y la predicción de un modelo.
- Gestores datos de entrada y salida de redes que facilitan la carga de ellos y la maximización del uso de memoria.

Así mismo los flujos de ejecución de cada uno son distintos, en un renderizador su ejecución se realiza sobre la resolución de la ecuación de render mediante un integrador, para ello el integrador aplica una técnica de aproximación para obtener la radiancia en cada píxel. En el caso de las librerías el flujo se centra en el entrenamiento, primero cargando los datos y después iterativamente la red va aprendiendo de ellos, pero si en vez de entrenar se evalúa una red solo es necesario cargar un único dato con el que se va a evaluar, esta evaluación se realiza mediante una única iteración de la red.

Debido a todo lo expuesto anteriormente no se posee una interfaz de contacto entre ambas partes que facilite trabajar juntos. Así mismo las redes pueden ser usadas de las dos formas anteriormente descritas lo cual dificulta aun más su uso de forma reutilizable. También se debe tener en cuenta que las redes poseen dos formas de ejecución las cuales, cambia su flujo de trabajo tal y como se ha comentado antes.

Otro aspecto importante que dificulta su uso conjunto son los tipos de datos. Cada sistema usa tipos distintos y estos no son equivalentes. Las librerías de aprendizaje usan los tensores anteriormente comentados como principal tipo de dato. Pero en un motor de render existen muchos más tipos, desde la forma interna de representación de imágenes o texturas hasta el formato de la radiancia que por ejemplo pueden ser tres canales rgb o un conjunto de espectros de la luz para una representación más precisa. Si se quisiera entrenar un modelo que fuera capaz de, a través de los canales. rgb aprender

a transformarlos en su espectro de luz, el modelo no podría recibir directamente los datos que se generaran en el renderizador dado que tienen formatos distintos, también seria necesario realizar el proceso de transformación de los datos de referencia, es decir, el espectro de luz para que el modelo pueda compararse y aprender. Este ejemplo ilustra como, para un problema sencillo requiere realizar dos transformaciones de datos, que en caso de querer usar el modelo entrenado como parte del renderizador habría que añadir más traducciones de tipos.

Como se ha comentado en esta sección los renderizadores y las librerías de aprendizaje automático son elementos disjuntos por defecto. Cada uno tiene sus peculiaridades individuales y carecen de algún tipo de interfaz común. Sin embargo ambos son dos sistemas que pueden ser muy útiles en conjunto, tal y como se ha comentado en la sección 2.2 ya se están utilizando de forma conjunta para la resolución de problemas muy variados. Por ello se ha desarrollado un sistema que integra ambas partes facilitando su uso y la interacción entre ellas. Para validar su utilidad se ha reproducido un artículo de investigación de render que aplica redes neuronales.

Capítulo 3

Arquitectura propuesta

En este capítulo se propone una arquitectura que integre las librerías seleccionadas y se presentan los motivos de su elección. Así mismo se explica qué patrones se utilizó para modelar el comportamiento del sistema implementado.

3.1. Estudio de los motores de render

Un motor de render es una librería desarrollado para la generación de imágenes sintéticas. En este trabajo solo se tendrán en cuenta aquellos que están basados en físicas. En la actualidad hay bastantes de ellos que son usados tanto a nivel de industria como de investigación. Los usados en la industria son para uso privado, como Hyperion de los estudios Disney o están a la venta bajo licencia, como Arnold de la empresa Autodesk. Debido al carácter privado de estos renderizadores o al precio que supondría adquirir una licencia que permita su uso y modificación, se decidió no hacer uso de ellos. Por otro lado existen diversos motores que sí son públicos y de código abierto como PBRT[15], Tungsten[16] o Mitsuba[17] todos comparten una estructura similar pero Mitsuba es el más utilizado actualmente en investigación.

Mitsuba es multiplataforma soportando tanto Windows, Linux y MacOs; además esta escrito en C++ y proporciona una limitada interfaz en *python*, posee una alta modularidad y proporciona una gran variedad de algoritmos de render ya implementados permitiendo realizar comparaciones entre distintos métodos de renderizado. Todos sus componentes, como cámaras, integradores, emisores y otros que implementan los elementos comentados en el capítulo anterior sección 2.3 y que son implementados como *plugins* que pueden ser cargados dinámicamente. Así como un sistema de control de concurrencia para ejecución de procesos paralelos. Por ejemplo, en algunos algoritmos de renderizaso como *path tracing* cada píxel de una imagen puede ser calculado de forma independiente lo que lo convierte en un proceso completamente paralelizable. Mitsuba también implementa un modo de

ejecución distribuida para generar imágenes en múltiples equipos aprovechando aún más la característica paralelizable del proceso de renderizado. Todo ello facilita la implementación de nuevos métodos sin necesidad de realizar grandes modificaciones en el resto del sistema.

Como se ha comentado Mitsuba es un renderizador muy desarrollado, que tiene una alta modularidad permitiendo ser modificado fácilmente y tiene una gran cantidad de algoritmos ya implementados. Además de ser el más usado en investigación. Por todo ellos se escogió este motor para trabajar con el.

3.2. Elección de la librería de aprendizaje automático

En la actualidad hay un gran número de librerías que permiten el uso de redes neuronales y otros métodos de aprendizaje automático de forma sencilla. Inicialmente se buscó si se había realizado alguna integración de una librería de aprendizaje con Mitsuba que permitiera estudiar otras aproximaciones. Se encontraron varios artículos de investigación como *Müller et al.*[9] y *Rainer et al.*[3] en el que se mencionan que se ha realizado algún tipo de integración pero no posee código libre. Para minimizar el coste de desarrollo de la interfaz se tomó la decisión de que la librería debería tener una interfaz en C++ como Mitsuba. Como se ha comentado antes, Mitsuba posee una interfaz en python pero se restringe a llamadas a funciones y métodos de objetos, no permite una manipulación directa de ellos, por lo tanto, se descartó dicho lenguaje.

Con la limitación de que la librería debía tener interfaz en C++ se procedió a la búsqueda de aquellas que poseyeran dicha interfaz además de ser multiplataforma y que su uso estuviera extendido tanto en industria como en investigación. Con estos requisitos se obtuvieron tres opciones: *LibTorch*, *TensorFlow* y *Caffe2*. Sobre ellos se realizó un estudio de sus opciones y capacidades para seleccionar el más conveniente.

- *TensorFlow*: Desarrollado por Google, con un gran recorrido, lleva desde 2015 siendo desarrollado. Su interfaz en C++ es relativamente reciente y no está muy desarrollada. Faltan muchas funciones por ser adaptadas, si bien para proyectos sencillos podría ser válida. En caso de querer implementar redes complejas la librería pronto demuestra no ser suficientemente avanzada al faltar le operaciones y capas por implementar.

- *Caffe2*: Desarrollado por Facebook y más orientado a producción. Posee una interfaz más avanzada que *TensorFlow* pero su uso es a bajo nivel, requiere tener un conocimiento más profundo sobre aprendizaje automático para su uso dado

que no ofrece una interfaz de alto nivel. Hacer uso de una capa de una red supone realizar muchas configuraciones que otras librerías esconden con interfaces más simples.

- *Libtorch*: Es una distribución de la librería *Pytorch* (desarrollada en python) diseñada para ser usada íntegramente en C++. Ha sido desarrollada por Facebook y tiene como base *Caffe2*. Proporciona una interfaz en alto nivel que sigue el esquema de la implementada originalmente en python. Permitiendo traducir código entre lenguajes con bastante sencillez. Esto permite aprovechar los desarrollos realizados por la comunidad que trabaja mayoritariamente en *Pytorch*. Así mismo posee implementadas múltiples redes muy usadas como ImageNet[18], DenseNet[19], ResNet[20] o GoogLeNet[21] permitiendo reutilizar los pesos de estas redes en nuevos modelos que se basen en ellos.

Durante unas prácticas realizadas los meses de julio y agosto, se desarrolló un trabajo previo a este proyecto se usó *Caffe2* por ser la librería más desarrollada de ese momento. *LibTorch* estaba en fase beta en por aquel entonces y no completamente desarrollada por lo que no se consideró su uso. Sin embargo, hacer uso de *Caffe2* en C++ supuso un gran reto debido al alto nivel de conocimientos que requería para programar redes neuronales. Su interfaz era de muy bajo nivel y se tuvo que hacer uso de interfaces diseñadas por la comunidad que aunque facilitaban el trabajo faltaban elementos de ser abstraídos a interfaces de alto nivel. Se realizaron pruebas de integración sencillas con el renderizador Mitsuba que resultaron en una fuerte incompatibilidad entre ellos, las librerias internas de cada uno colisionaban constantemente dando errores de compilación. Por ello cuando se inicio este trabajo *LibTorch* ya había sido publicada oficialmente y se probó en una primera instancia para ver si provocaba el mismo tipo de problemas que con *Caffe2*. Sin embargo, la librería resultó ser mucho más sencilla de usar debido a su interfaz de alto nivel y su integración dio problemas de colisiones con Mitsuba.

3.3. Diseño propuesto

Como se ha comentado en la seccion 2.3 existe una gran problemática en el uso de ambos sistemas de forma conjunta, dado que están diseñados para ser ejecutados de forma independiente y tanto sus flujos de ejecución como sus tipos de datos son muy distintos. Por ello se diseñó un sistema que permita su uso conjunto y su manipulación de una forma sencilla. Para cada uno de los problemas que se han abordado se les dio una solución mediante la implementación de un módulo. A continuación se detallan

que módulos se han implementados y que problema solucionan.

3.3.1. Mediador

Tanto Mitsuba como *Libtorch* poseen interfaces demasiado distintas como para ser usadas directamente entre ellas. Además existe la problemática de que al seguir siendo desarrolladas sus métodos pueden sufrir cambios que en un futuro provoquen que sea necesario modificar parte de la integración desarrollada. Debido a ésto se decidió mantener ambas partes por separado para aislarlas y generar una interfaz de comunicación entre ellas. Para su desarrollo se tuvo en cuenta las dos formas en las que se puede usar un modelo de red neuronal.

- Evaluación: En este proceso se hace uso de un modelo pre-entrenado de red neuronal al que se pasan como entrada un conjunto de datos para obtener una predicción. En este contexto Mitsuba es quien quiere realizar la evaluación y para ello solicita al mediador la evaluación de la red, el proceso entero se ha descrito en la figura 3.1 y se puede observar un diagrama de comunicación entre las partes y como el mediador mantiene asilado a *LibTorch* de Mitsuba. De esta forma se consigue que Mitsuba sea agnóstico a la implementación de la red y solo deba pasar al mediador el tipo de red a evaluar y los datos.
- Entrenamiento: Este es un proceso más complejo y es equivalente al explicado en la sección 2.2. Dado que la red debe ser abastecida con un conjunto de datos con los que itera aprendiendo de ellos, se desea que Mitsuba sea la encargada de generar dichos datos de entrenamiento permitiendo que sean servidos en tiempo real. Por ello la generación debe hacerse de forma paralela al entrenamiento. En la figura 3.2 se describe en la parte superior este proceso de generación. Por ilustrar el trabajo que se realiza en el siguiente capítulo se indica en él, que el mediador ordena renderizar un dato, sin embargo éste puede ser de cualquier tipo, como por ejemplo, valores de una textura o ángulos de incidencia de la luz en un punto y la cantidad que refleja. En la parte inferior de la figura 3.2 puede verse como el modelo solicita datos al mediador para realizar una iteración del entrenamiento.

Para diseñar el mediador se hizo uso del patrón de diseño *mediator* su objetivo es encapsular la interacción entre múltiples objetos. Este patrón modela exactamente lo que se desea hacer entre las dos partes, la encapsulación de su interacción y separación. Así mismo este módulo debe ser usado por varios procesos de forma simultánea y como se quiere que todos accedan a la misma instancia de él, se ha modelado siguiendo otro patrón de diseño denominado *singleton* que fuerza a que solo exista esa instancia mediante la imposibilidad de ser creado más de una vez.

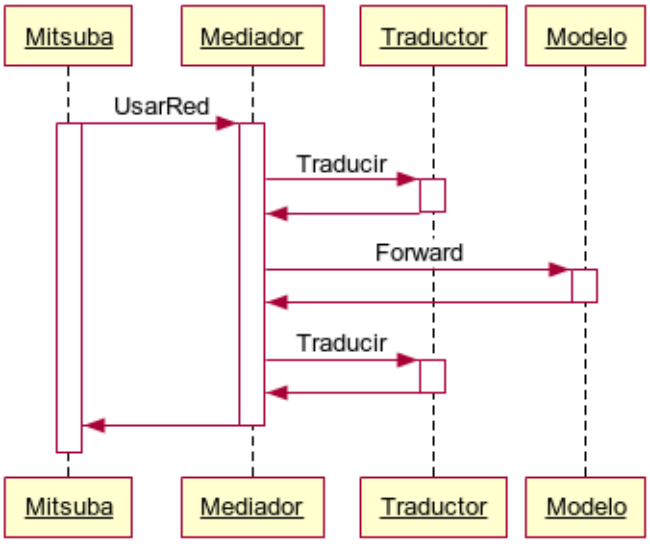

Figura 3.1: Diagrama de secuencia del uso de una red

3.3.2. Buffer

Como se ha comentado en la sección 3.3.1, ambos sistemas deben ser capaces de transferir información en la etapa de entrenamiento. Sin embargo ambos poseen tiempos de ejecución distintos y si operasen secuencialmente, ello supondría una perdida de tiempo y de capacidad. Para ello se diseñó un buffer que almacena los datos generados para que las redes puedan solicitarlos y usarlos, en la figura 3.2 puede verse como es usado en ambos casos para realizar la transferencia de datos, de esta forma con el buffer se elimina la dependencia directa entre el generador y las redes. Además para facilitar las posibilidades de uso del sistema se desarrolló para ser usado por múltiples redes simultáneamente, haciendo uso todas ellas de los mismos datos de entrenamiento. Esto facilitará realizar pruebas de distintos parámetros de entrenamiento o distintas arquitecturas de red sin tener que realizar modificaciones en el futuro.

Esta utilización concurrente del buffer por parte de distintos elementos(hilos de ejecución) genera un problema de acceso a un elemento compartido, el cual se produce al realizar lecturas y escrituras al mismo tiempo lo cual puede conllevar una corrupción de los datos. Para asegurar su correcto funcionamiento se le añadió un monitor; este elemento de control de concurrencia asegura el acceso en exclusión mutua a los datos y además permite la espera de las redes hasta que el buffer dispone de los datos solicitados.

3.3.3. Traductor

Para poder hacer uso de todo este sistema, hace falta un elemento que permita entenderse entre Mitsuba y *LibTorch*. Puesto que los tipos de datos de cada uno son

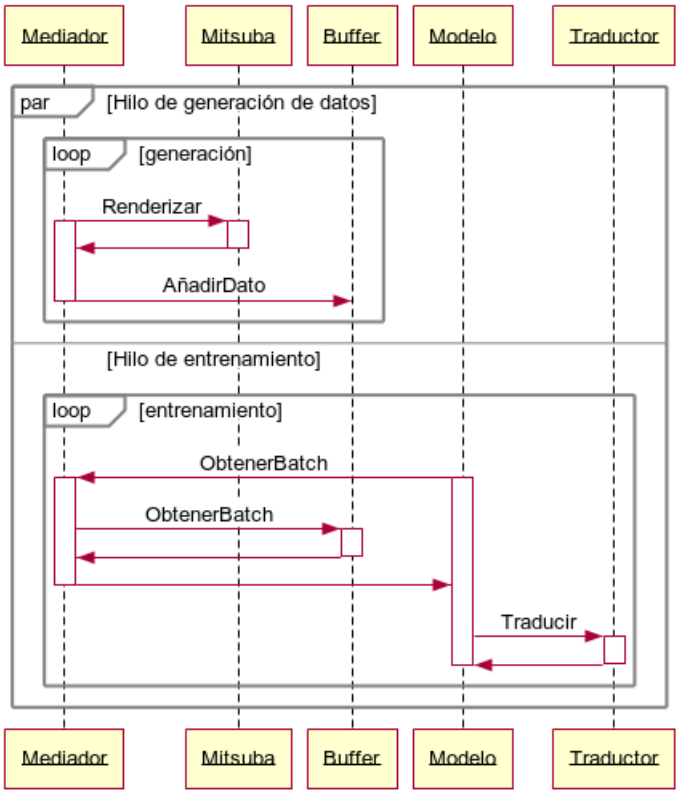

Figura 3.2: Diagrama de secuencia de entrenamiento de una red y la generación de datos

distintos, se desarrolló un traductor con interfaz genérica, de esta forma las llamadas son iguales sin importar el tipo de dato que se traduzca, ésto permite dar más flexibilidad al uso del sistema al no depender de funciones con signaturas especificas. Es necesario implementar una nueva función para cada tipo de dato pero es el compilador quien infiere cual se usa.

3.3.4. Esquema global

Para ver el esquema global se ha esquematizado como queda el sistema completo y que dependencias existen. En la figura 3.3 puede verse como tanto el mediador que contiene al buffer como el traductor dependen de Mitsuba y de *LibTorch*. Se puede observar que no hay dependencia directa entre ninguna de las dos partes habiendo conseguido con ello el objetivo que se deseaba de aislarlas.

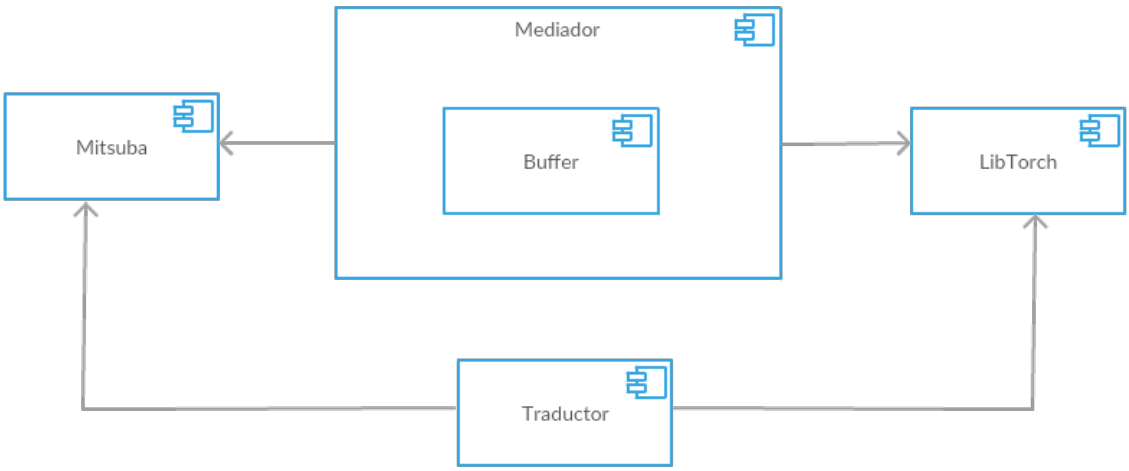

Figura 3.3: Diagrama de componentes del sistema desarrollado

Capítulo 4

Evaluación del sistema

En este capítulo se procede a evaluar y demostrar la utilidad del sistema implementado y descrito en el capítulo 3. Para ello se optó por reproducir un artículo de investigación para poder usar el sistema sobre un problema real. El tipo de problema escogido fue de eliminación de varianza mediante redes neuronales sobre imágenes renderizadas. Como se comentó en el marco teórico, renderizar imágenes se hace mediante la resolución de la ecuación de render (sección 2.1.1), siendo esta imposible de resolver analíticamente se deben hacer uso de técnicas de aproximación como Monte Carlo (sección 2.1.3) el cual aproxima una función mediante un muestreo aleatorio. Sin embargo esta técnica genera una varianza que para ser reducida a la mitad se debe elevar al cuadrado el número de muestras. Esto supone que la generación de imágenes libres de varianza requiere un elevado numero de muestras. En la figura 4.1 se pueden observar una imagen generada con distinta cantidad de muestras y como conforme se va aumentando su cantidad la varianza disminuye.

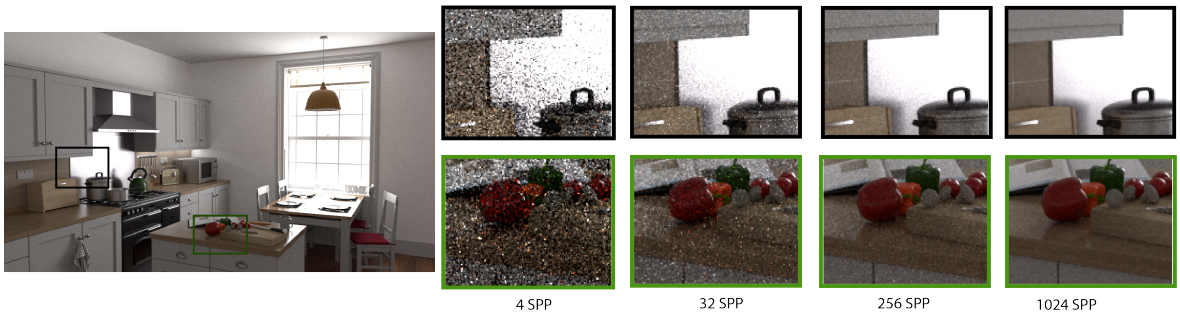

Figura 4.1: Ejemplos de varianza con diferente número de muestras por píxel (SPP)

Este problema es muy investigado y de particular interés para las empresas que trabajan en la generación de imágenes por ordenador y que, con un buen sistema de eliminación de varianza se pueden conseguir imágenes libres de ella con un número significativamente reducido de muestras.

Se estudiaron distintos artículos del estado del arte sobre eliminación de varianza

sobre render tales como *Bako et al.* [10] aplicado sobre imágenes estáticas y *Chaitanya et al.[11]*, *Vogels et al. [22]* aplicados sobre un conjunto de renders de vídeo. Finalmente se escogió el articulo a implementar de *Bako et al.* [10] que a diferencia de los otros, trabaja con una única imagen pudiendo aprovechar los recursos de la comunidad como escenas públicas. A continuación se detallan las partes más importantes del artículo y posteriormente se presentaran las adaptaciones realizadas sobre el mismo, para realizar su implementación sobre el sistema desarrollado en este trabajo. Para finalizar se explicará que variaciones se han implementado sobre el método del artículo y se mostraran y discutirán los resultados obtenidos.

4.1. Resumen del artículo

Como hemos comentado anteriormente el trabajo de *Bako et al.* [10] se centra en la eliminación de varianza mediante el uso de redes neuronales. Si bien el uso de redes para la resolución de este problema no es nuevo, en otros trabajos como el de *Chaitanya et al.[11]* cuya red tiene como salida directamente una imagen sin varianza. Sin embargo, en este artículo introducen un nuevo método en el que obtienen en su lugar una serie de kernels que se aplican sobre la imagen. Obteniendo un kernel por píxel de la imagen original estos, son aplicados sobre el vecindario centrado en dicho píxel. La red codifica en los kernel una distribución de energía en función de los valores de ese vecindario que al aplicarse se obtiene un valor más preciso y mas cercano al que tendría si careciera de varianza. La idea de esto es que, al ser un elemento flexible el kernel puede adaptarse a las peculiaridades de cada zona, un ejemplo claro es si se tiene una zona en la que haya un borde o un cambio de iluminación muy brusco el kernel debe adaptarse para no verse afectado por los píxeles vecinos que sean muy distintos al que se esta aplicando el kernel.

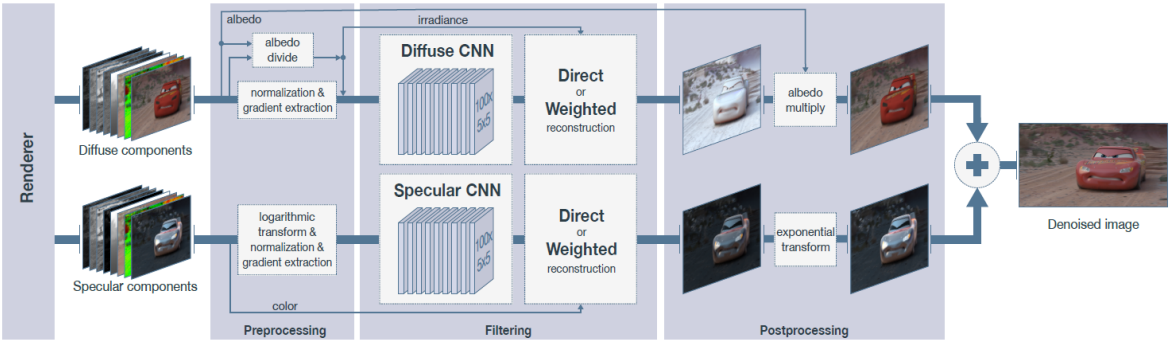

Figura 4.2: Arquitectura de la red. Imagen de *Bako et al.*[10]

Pese a que este tipo de técnica podría usarse con una entrada directa de la imagen

a la que se aplica la eliminación de varianza. En el artículo se demuestra que si se separa la iluminación que hay en ella entre sus componentes especular y difusa permite entrenar dos redes por separado, obteniendo un mejor resultado. La idea de esta separación radica en la diferencia de varianza que se genera por la forma en que un material refleja parte de la radiancia que incide en él. Como se ha explicado en la sección 2.1.1 del marco teórico un material puede reflejar de distintas formas según su BRDF, si es difuso refleja en todas las direcciones por igual, pero si es especular refleja más en direcciones concretas. Esta reflexión genera un tipo de varianza distinta para cada uno, alta frecuencia en el caso de especulares y baja frecuencia para los difusos. Por tanto esta separación permite que cada una de las dos redes pueda especializarse en un tipo de varianza sin verse contaminada por la frecuencia que generan los materiales de otro tipo. En la figura 4.2 puede observarse un esquema de la implementación realizada en el artículo y como una vez las dos redes han sido ejecutadas, la imagen global es recompuesta.

$$\tilde{p}_{difuso} = p_{difuso} \oslash (f_{albedo} + \epsilon) \tag{4.1}$$

$$\tilde{p}_{especular} = \log(p_{especular} + 1) \tag{4.2}$$

$$(\tilde{\sigma}_{difuso})^2 \approx \sigma^2_{difuso} \oslash (f_{albedo} + \epsilon)^2 \tag{4.3}$$

$$(\tilde{\sigma}_{especular})^2 \approx \sigma^2_{especular} \oslash (\tilde{\sigma}_{especular})^2 \tag{4.4}$$

$$\mathrm{x} = \{\tilde{c}, G_x(\{\tilde{c}, f\}), G_y(\{\tilde{c}, f\}), \tilde{\sigma}^2, \sigma_f^2\} \tag{4.5}$$

En la figura 4.2 puede verse que la entrada de cada red son un conjunto de imágenes aparte de la que contiene la componente difusa o especular según corresponda, a las que llamaremos imagen principal. Esta serie de imágenes extra son: varianza de la imagen principal, mapa de normales, mapa de albedo (color del material de objeto sin calcular iluminación) y mapa de profundidad de la escena además de sus respectivas varianzas, En la figura 4.3 puede verse un ejemplo de cada una de ellas. Sin embargo estas imágenes no son utilizadas directamente en la red. Se les realiza un preprocesado de forma que se generan para cada una de ellas sus gradientes sobre sus ejes X e Y. Además a las principales. y sus varianzas se les aplica previamente una transformación siguiendo las ecuaciones 4.1, 4.2, 4.3 y 4.4. Quedando la entrada finalmente con el

conjunto de la ecuación 4.5 siendo $\tilde{c}$ y $\tilde{\sigma}^2$ las imágenes especular o difusa y su varianza según corresponda. La idea detrás de aportar este conjunto es aprovechar toda la información que se puede obtener de un renderizador y que ayudan a describir la escena. Con ello se busca que la red pueda encontrar patrones y estructuras entre la varianza de la iluminación y la información que se le aporta. En el artículo se demuestra como esta información es aprovechada correctamente por la red dando mejores resultados que sin ella.

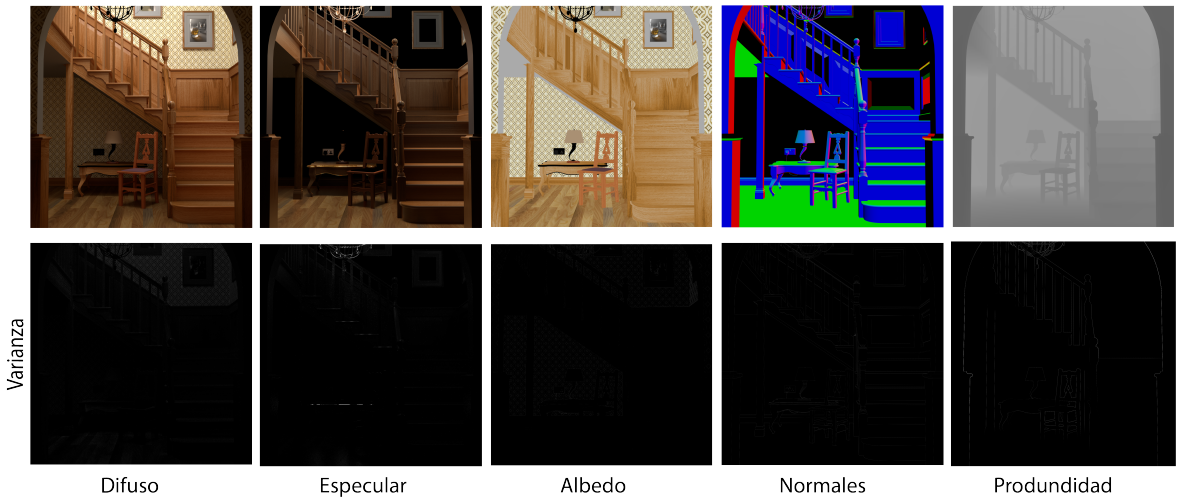

Figura 4.3: Imagen generada para el entrenamiento con cada una de sus componentes.

El entrenamiento de ambas redes se realiza de forma separada, y una vez han convergido se procede a realizar un proceso de *fine-tuning* o ajuste, en el que se entrenan de nuevo de forma simultánea y tomando la imagen final después de ser unida como objetivo a ajustar. Esto elimina pequeños artefactos que se generan al unir ambas partes y que cada red por separado no había podido eliminar.

4.2. Adaptación y cambios sobre el artículo

Para replicar este artículo se optó por modificar y simplificar algunas partes para centrar el trabajo en la verificación del sistema implementado.

Primeramente se realizó la modificación del integrador que implementa el algoritmo de renderizado *path tracing* de forma que éste genere la separación de iluminación. En el artículo se realiza implementando el método de *Zimmer et al.* [23] que separa la iluminación conforme se va generando el camino de randiancia del algoritmo *path tracing*. Dado que el objetivo de este trabajo no es la reproducción exacta del método, se decidió que la separación se simplificaría e implementaría de la siguiente forma: en lugar de tener componentes especular y difusa en un mismo camino, éste se considera

que contiene únicamente como una de las dos componentes en función de la forma en la que se realiza el primer rebote del camino, por ejemplo si el primer rebote es sobre un objeto y se decide que este se calcula como un elemento difuso entonces todo el camino que se genera sera considerado así. Este cambio respecto al método original genera en la separación diferencias con las imágenes que proporciona el artículo. En la figura 4.4 se puede observar una comparación de una imagen generada de ambas formas y como el método implementado en este trabajo tiene problemas con los reflejos especulares en espejos y otros objetos muy especulares debido a que ellos reflejan la casi totalidad de la radiancia sin alterarla. Si se observa el espejo o el grifo, se puede ver como en la imagen izquierda contiene el reflejo de material difuso.

Figura 4.4: Imagen de la componente especular, izq: Imagen generada según el tipo del primer rebote, dcha: Imagen proporcionada en el material extra de *Bako et al.*[10]

Uno de los efectos de la separación de iluminación es la generación de zonas sin información en la escena que quedan completamente negras, esto ocurre por ejemplo si una pared es completamente difusa sin componentes especulares, en la imagen generada con la radiancia especular se observará negra. Estas zonas son descartadas al realizar el entrenamiento. Teniendo un total de 12 imágenes para entrenar y 2 para validar (en la figura 4.5 pueden verse todas ellas), con una calidad de 128 muestras como entrada y 1024 muestras como imagen de referencia. Se toman 1000 parches de cada una aleatoriamente de los cuales, se eliminan los que resultaron completamente negros. Como se obtuvo inicialmente un número relativamente corto de parches se decidió

aplicar técnicas de aumentación de datos como la rotación o el volteo de los parches, obteniendo una media de 36000 parches. Cabe destacar que en el artículo hacen uso de un total de 600 imágenes de las que sacan 400 parches por cada una de ellas. Esto es una diferencia notable entre ambos bancos de datos.

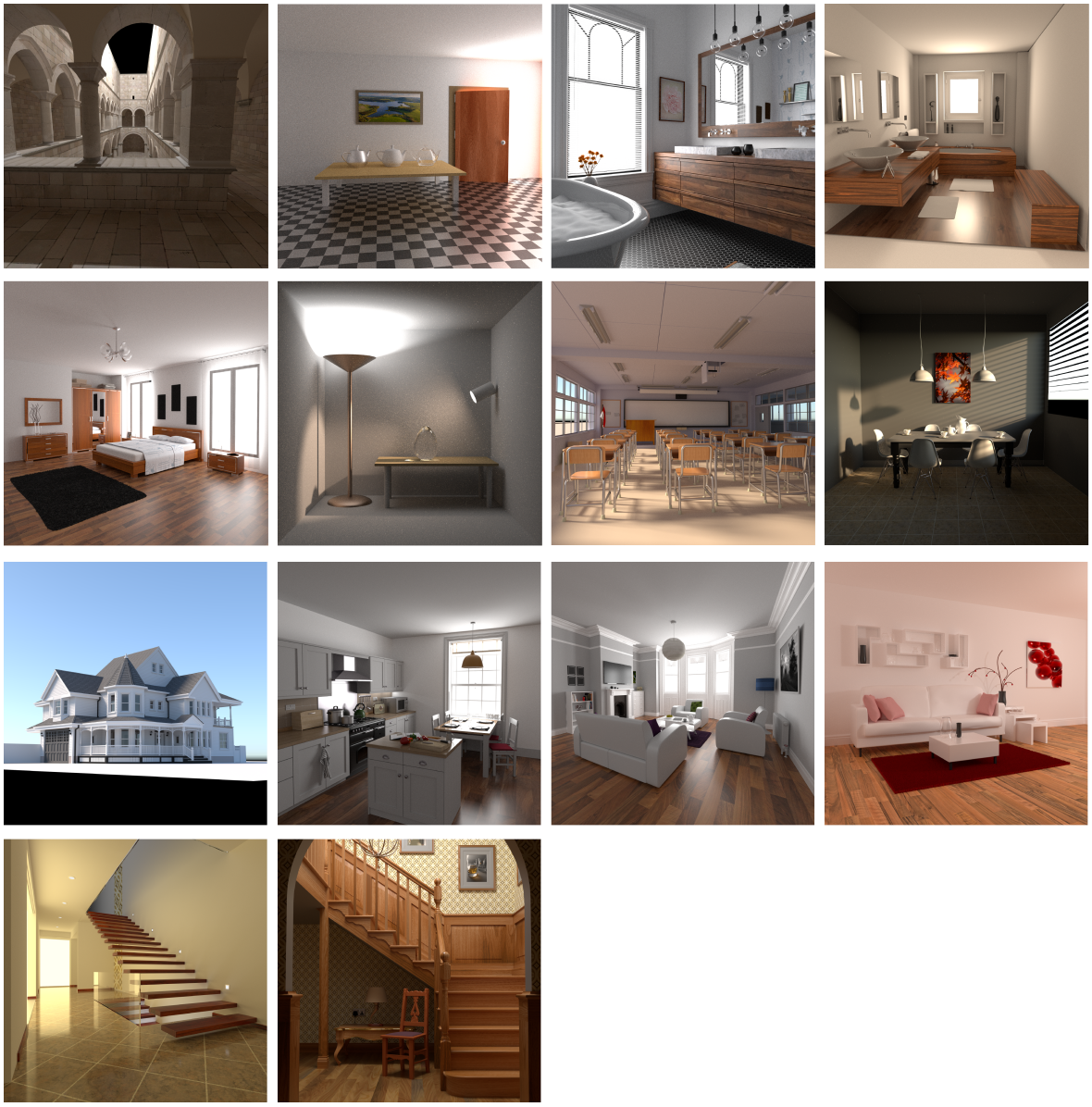

Figura 4.5: Imágenes generadas para el entrenamiento siendo la primera y quinta usadas para validación [16][17]

Para realizar el entrenamiento se hizo uso del sistema desarrollado en el capítulo 3 implementando todo lo descrito anteriormente junto con las dos redes en él. De esta forma mientras el sistema genera los datos, las redes los solicitan y se puede tener ambos trabajos de forma simultánea, permitiendo tener un flujo te trabajo único que automatiza la transferencia de datos entre Mitsuba y *LitTorch*.

Tras implementar la adaptación del artículo y proceder a entrenar las redes, se

observó como no convergían a soluciones óptimas. El tipo de kernels que aprendían pueden verse en la figura 4.6 en los que cada uno tienen forma de gaussiana, estando todos ellos centrados y conforme se va alejando del centro del kernel pierden intensidad. Una de las primeras conclusiones a las que se llegó fue, que aunque el resultado no era bueno si que tenia sentido que aprendiera de esa forma, al aplicarse un kernel de este tipo se da más peso a los píxeles mas cercanos al central, teniendo en cuenta suelen compartir características y a los mas alejados les da menos peso. Sin embargo este tipo de kernels tienen el efecto de emborronar la imagen. Para comprobar si la hipótesis era correcta se ejecutó la red que proporciona el artículo como material suplementario. Los kernels obtenidos se pueden ver en la figura 4.7 y como estos varían de unos a otros manteniendo una estructura para poder preservar los detalles de la imagen.

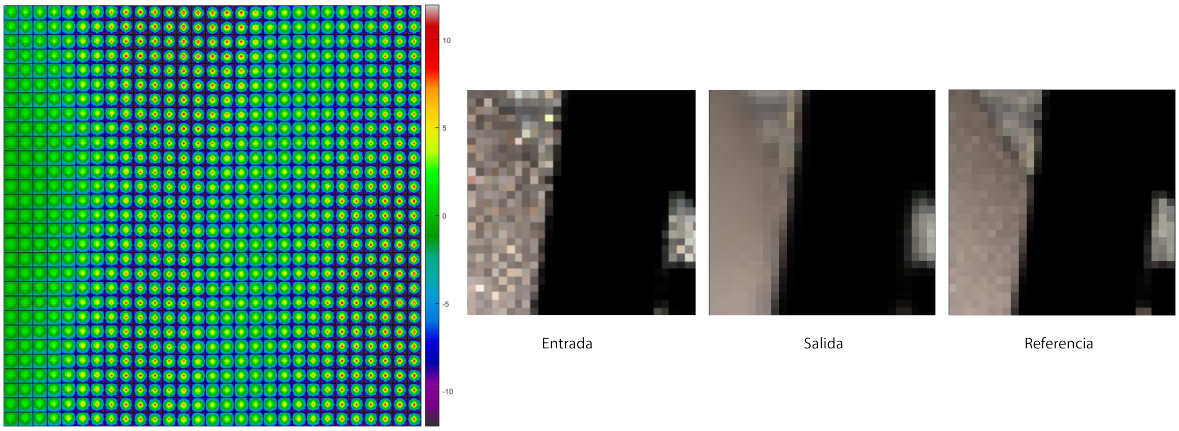

Figura 4.6: Kernels obtenidos en entrenamientos sin convergencia para el parche de la izquierda, cada uno posee forma de gaussiana. Se muestran un total de 29x29 kernels con un tamaño cada uno de 21x21 píxeles.

Para buscar un mejor resultado se hicieron entrenamiento con distintos parámetros de la red. Aprovechando una de las características del sistema implementado el cual permite usar múltiples redes de forma simultanea con los mismos datos, lo que permite por ejemplo, probar para una misma red diferentes instancias de ella con distintos parámetros. Finalmente se consiguió una mejor convergencia en la red difusa al dividir para el máximo valor de cada parche. En la figura 4.8 puede observarse como finalmente los kernels adquieren una forma distinta al inicial de la figura 4.6 y poseen una estructura similar a la obtenida en los kernels de la figura 4.7. En la sección 4.4 se describen los resultados obtenidos y se hace una valoración sobre ellos.

4.3. Nuevas aproximaciones

Como variación sobre el artículo se propuso aplicar el método a imágenes con medios participativos, estos son descritos en la sección 2.1.2. El medio configurado es isótropo,

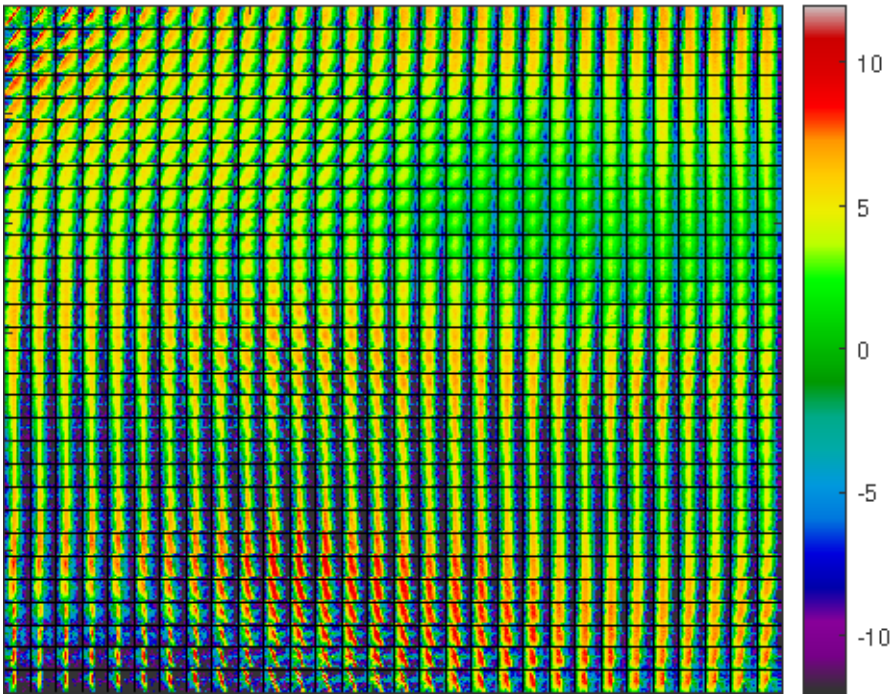

Figura 4.7: Kernels obtenidos de la red proporcionada en el artículo

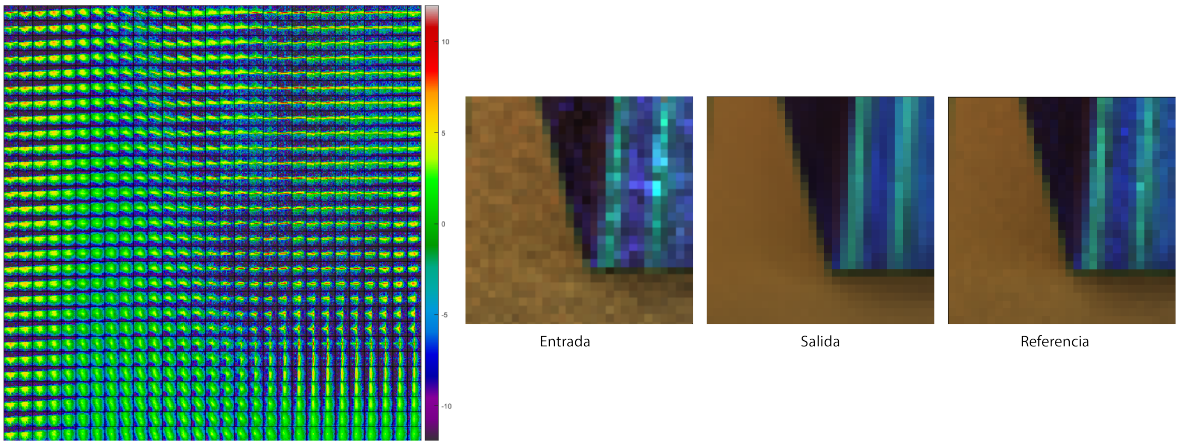

Figura 4.8: Kernels obtenidos en entrenamiento convergido, obtenido para el parche de la derecha, cada uno de ellos es distinto se muestra en la izquierda los kernels obtenidos para el parche de la derecha

homogéneo y posee coeficientes de absorción y dispersión iguales, lo cual implica que las partículas tienen la misma probabilidad de dispersar la luz o de absorberla. además las partículas están distribuidas por toda la escena de forma homogénea.

Siguiendo la aproximación del artículo se generaron imágenes en las que se separó la iluminación de los medios participativos del resto de componentes, de la misma manera que se hizo al separar la radiancia difusa o especular. Cuando el primer rebote es sobre un medio, el camino que se genere desde él se considera como componente de medios. En la figura 4.9 se pueden ver una dos las imágenes generadas y como conforme los objetos están más lejos de las fuentes de luz son menos iluminados.

Figura 4.9: Imagen con los medios separados

Como en la sección anterior se implementó esta parte en el sistema desarrollado para poder generar los datos de forma simultánea al entrenamiento. Las imágenes que se generaron fueron de la misma calidad que en el caso anterior, 128 muestras para entrenar y 1024 como imagen de referencia, en la figura 4.10 puede verse un ejemplo de un parche de entrenamiento y como la de 128 posee una varianza muy alta comparada con la de 1024. El entrenamiento con estos datos no convergió y al analizar los datos se observó la poca definición que poseían, por lo que se decidió que la mejor opción era generar imágenes de más calidad con 8 veces más muestras. Sin embargo ese tipo de imágenes requiere una cantidad de tiempo 10 veces mayor, unas 20 horas de media para una imagen de 2048x2048 de resolución sobre un equipo con 36 hilos de ejecución. Generar todo el banco de datos requeriría un mínimo de 10 días .

En el sistema desarrollado cuando se entrena (parte superior de la figura 3.2, sección 3.3) se ordena a Mitsuba que genere imágenes mientras las redes entrenan, como en la sección anterior las imágenes se generaban enteras y luego eran troceadas para ser

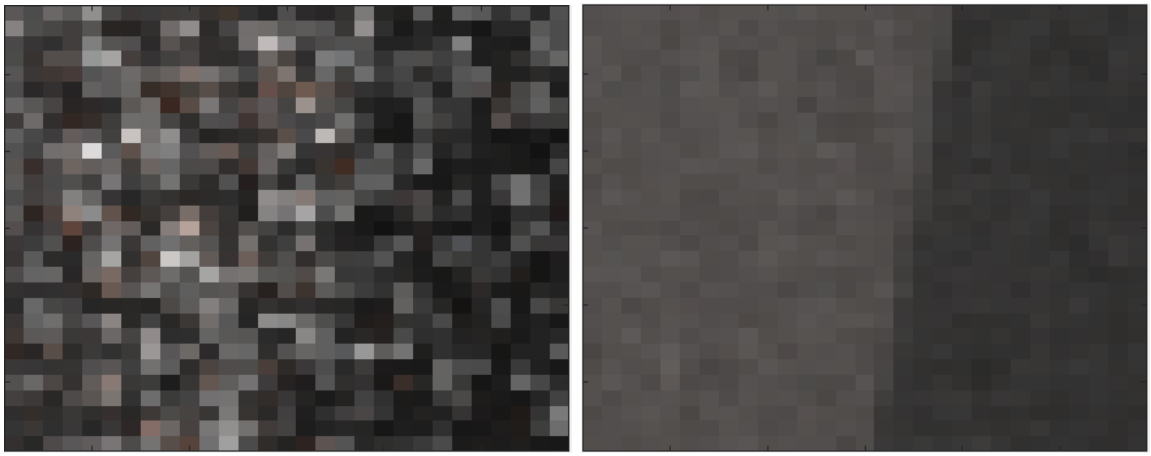

Figura 4.10: Parches inicialmente generados, izq: 128, dcha: 1024

servidas a la red. Sin embargo esto no era viable en medios debido al alto tiempo que requería generar una sola imagen. Por ello se decidió modificar dicho bucle para generar parches pequeños del tamaño que toma la red, de forma que se puedan usar inmediatamente. El objetivo de esto es poder ver la evolución de la red conforme va entrenando, dicho entrenamiento cuando es correcto produce un descenso del error, el cual se produce principalmente en las primeras etapas. Por tanto con el sistema desarrollado podemos, con una parte muy pequeña de los datos generados, ver si se debe continuar el entrenamiento o hay que detenerlo para probar con otros parámetros. Esto permitió hacer múltiples pruebas sin necesidad de tener el conjunto total de los datos generados, ahorrando horas de trabajo y espera.

4.4. Resultados obtenidos

Para el sistema de eliminación de varianza en imágenes difusas se obtuvieron resultados significativamente buenos, el modelo es capaz de eliminar la varianza y conservar las formas de los objetos sin emborronarlos, esto es un elemento importante dado que se esta aplicando un kernel y si no esta bien calculado (como se ha explicado en la sección 4.2) puede provocar la pérdida de las formas, este efecto era el observado en entrenamientos que no convergían correctamente. En la figura 4.11 se puede ver como mantiene no sólo lo anteriormente mencionado sino también las texturas. El modelo posee un mayor acierto en las zonas con textura que en las lisas, donde se puede observar una mayor variación debido a que es más difícil esconder las imperfecciones. También se observó que el resultado final es mejor en ciertas situaciones que las imágenes generadas como referencia de la red.

Para los especulares el rendimiento es ligeramente inferior a la de los difusos. En la figura 4.12 se observa cierto emborronamiento en las partes lisas y sin bordes, por

Figura 4.11: De izquierda a derecha: Entrada 128 SPP, Salida de la red, imagen de referencia 1024 SPP

el contrario los bordes han sido mantenidos perfectamente por lo que la red ha sido capaz de aprender a identificarlos y respetarlos. Esto es un elemento importante dado que un método de eliminación de varianza debe sobre todo preservar los bordes ya que son elementos que en caso de ser emborronados son rápidamente identificables. Esto se debe en parte a que las imágenes no poseen una separación exacta del brillo especular y hay zonas como espejos, que tienen elementos de la radiancia difusa, en la figura 4.4 de la sección 4.2 se puede observar este efecto en el espejo de la imagen izquierda tal y como se ha comentado en dicha sección. Esto deriva en una perdida de las capacidades de la red para aprender a eliminar la varianza en las imágenes generadas al sufrir contaminación de radiancia difusa.

Con los medios participativos por el contrario, no se obtuvo resultados satisfactorios. En la figura 4.13 puede observarse como la salida de la red es un conjunto de formas que se repiten a lo largo de toda la imagen, si bien estos son capaces de mantener cierta coherencia en las formas. Esta figura que se repite es resultado del poco ajuste que ha realizado la red a los datos de entrenamiento. Para buscar una correcta convergencia

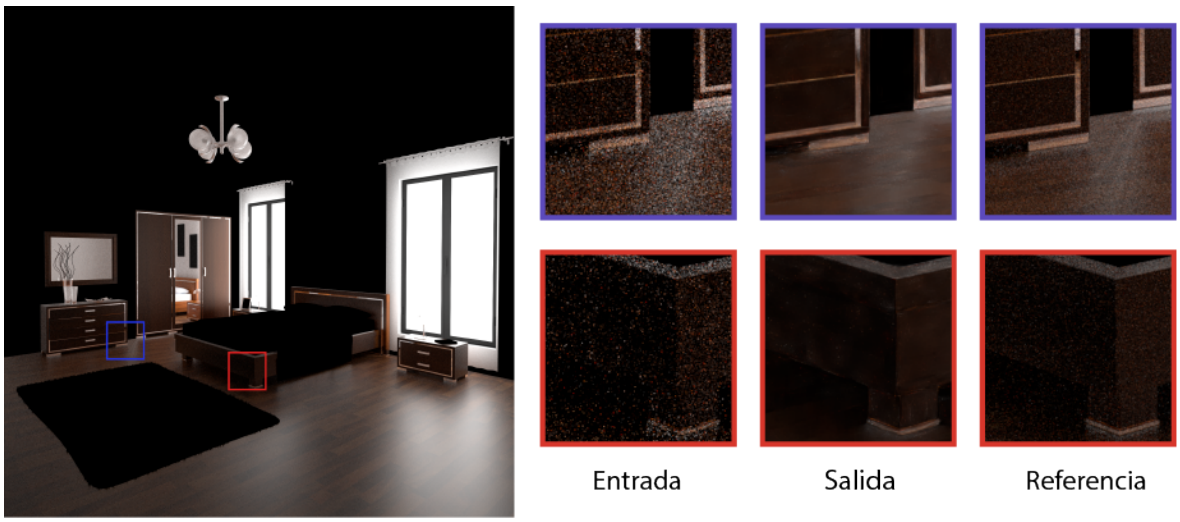

Figura 4.12: De izquierda a derecha: Entrada 128 SPP, Salida de la red, imagen de referencia 1024 SPP

se hicieron varias pruebas, tales como modificar los parámetros de la red o eliminar la imagen de apoyo que contiene el albedo que en el caso de los medios participativos no se ven afectados por ello al ser un medio que carece de él. Con las pruebas realizadas se llegó a la conclusión de que la eliminación de varianza en medios participativos debe ser reanalizada, debido a que su estructura es distinta de la que se genera en imágenes sin medios y la aproximación realizada no permite la convergencia a una solución correcta. Si bien se cree que el método es valido, se deben escoger otras formas de proporcionar información de apoyo a la red.

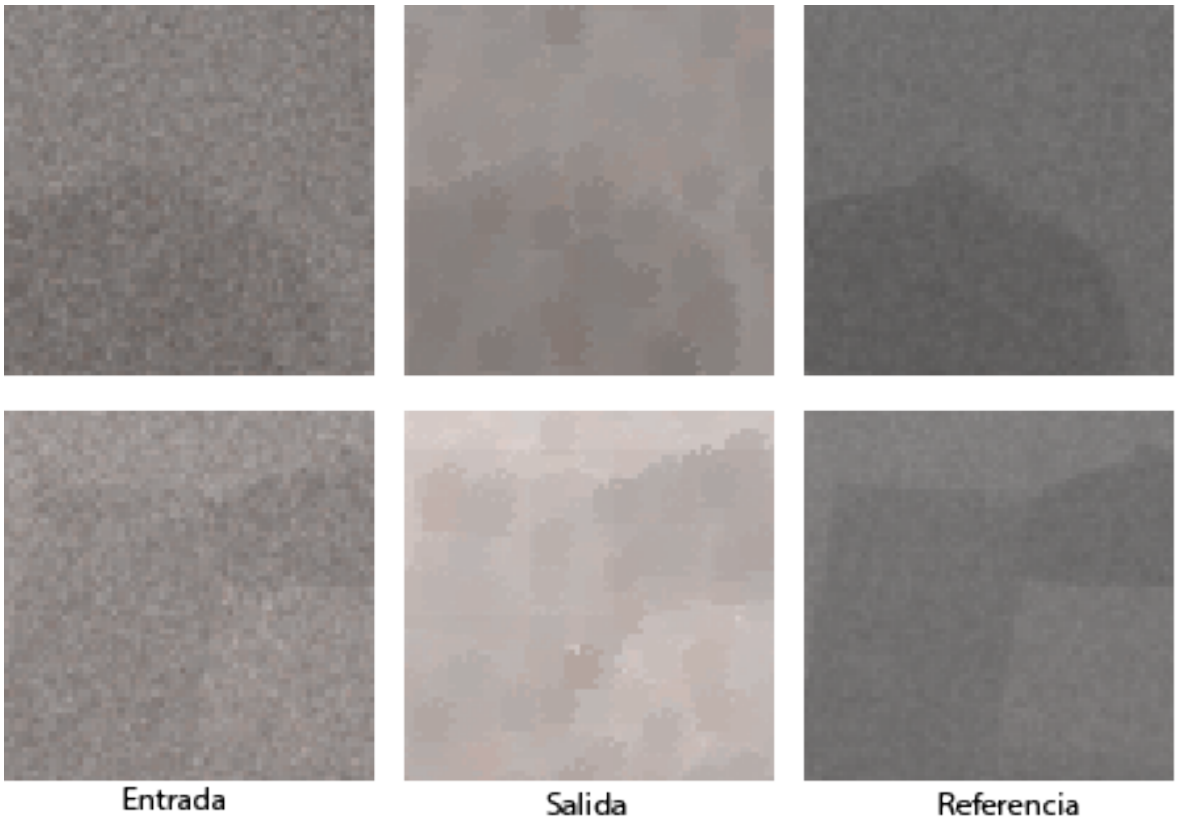

Figura 4.13: De izquierda a derecha: Entrada 1024 SPP, Salida de la red, imagen de referencia 8196 SPP

Capítulo 5

Conclusiones

El objetivo de este trabajo es diseñar e implementar una herramienta que permita hacer uso de redes neuronales directamente sobre un motor de render. De esta forma se unen dos sistemas creados independientemente en uno solo. Se realizó un estudio de que motores de render y librerías de aprendizaje había disponibles y de ellas se seleccionó el motor de render Mitsuba y la librería *LibTorch*. Su integración se realizó mediante el diseño e implementación de un módulo que media entre las dos, aislándolas entre sí y dando una interfaz adaptada para cada una.

Para validar el sistema desarrollado se propuso reproducir un artículo de investigación sobre eliminación de varianza en render mediante redes neuronales. Se realizó un estudio sobre el estado del arte seleccionando finalmente el artículo de *Bako et al.* [10]. Mediante su reproducción se probó la utilidad del mismo al poder realizar entrenamientos y generaciones de datos de forma concurrente consiguiendo ahorrar tiempo al unir los flujos de cada parte que sin el sistema desarrollado deberían ser ejecutados por separado y de forma manual. Durante la extensión del método del artículo a imágenes con medios participativos se pudo observar como en los casos en los que la generación de imágenes requería tiempos elevados se puede aprovechar más aún el sistema desarrollado al poder trabajar con partes de imágenes directamente, y no tener que esperar a su generación completa. Por ello se considera que el sistema desarrollado es lo suficientemente flexible y útil para poder adaptarse a diferentes escenarios.

Durante la finalización de este trabajo se tuvo conocimiento de que el creador de Mitsuba esta desarrollando una nueva versión diferenciable en tiempo de ejecución. Esto implica que todas las operaciones que se realizan guardan su derivada. Esto mismo es lo que realizan las librerías de aprendizaje automático que en base son sistemas de cálculo numérico diferenciales y es la base que permite hacer uso de redes neuronales y optimizarlas. Se desconoce hasta que punto se ha implementado tal diferenciabilidad pero el hecho de que el propio desarrollador avance en la dirección de una unificación entre ambos mundos apoya la utilidad de este sistema.

5.1. Trabajo futuro

Como trabajo futuro se considera interesante continuar con el desarrollo del sistema dada la utilidad que ha demostrado así como la continuación del estudio sobre medios participativos con objetivo de realizar una posible publicación. Otro punto sobre el que se desea trabajar es con la segunda versión de Mitsuba. Su diferenciabilidad permitiría realizar una integración más profunda, tomando como partida el sistema desarrolado en este trabajo, entre *LibTorch* y Mitsuba aumentando sus posibilidades de uso.

Capítulo 6

Bibliografía

[1] Disney. Frozen, 2013.

[2] Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans. Graph. (Proc. of Siggraph Asia)*, 36(6), November 2017.

[3] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural btf compression and interpolation. *Computer Graphics Forum (Proceedings of Eurographics)*, 38(2), March 2019.

[4] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[5] Subrahmanyan Chandrasekhar. Radiative transfer. 1960.

[6] Eric Veach. *Robust Monte Carlo methods for light transport simulation*, volume 1610. Stanford University PhD thesis, 1997.

[7] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[8] Delio Vicini, Vladlen Koltun, and Wenzel Jakob. Radiative transfer. 2019.

[9] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Nov'ak. Neural importance sampling. *arXiv preprint arXiv:1808.03856*, 2018.

[10] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017)*, 36(4), July 2017.

[11] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36(4):98:1–98:12, July 2017.

[12] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[15] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, October 2016.

[16] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/.

[17] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[19] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[22] Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4):124:1–124:15, 2018.

[23] Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum (Proceedings of EGSR)*, 34(4), jun 2015.

Lista de Figuras